China-pub.com
下载

第1章 系统结构

1.1 系统概述

Linux内核本身作为一个独立的东西是没有什么用的；它只是参与了一个更大的系统，成为那个系统的一部分，而该系统从整体上看是非常有用的。因此，在整个系统的上下文中介绍内核的作用就显得很有意义了。图 4-1-1显示了整个Linux操作系统的结构分析。

Linux操作系统由4个主要的子系统所组成：

- 用户应用程序 —— 在某个特定的Linux系统上运行的应用程序集合，它将随着该计算机系统的用途不同而有所变化，但一般会包括文字处理应用程序和 Web浏览器。
- O/S服务 —— 这些服务一般认为是操作系统的一部分（开窗系统，命令外壳程序，等等）；此外，内核的编程接口（编译工 具和库）也属于这个子系统。
- Linux内核 —— 这是本文的关注焦点；包括内核抽象和对硬件资源（如CPU）的间接访问。
- 硬件控制器 —— 这个子系统包含在Linux实现中所有可能的物理设备，例如，CPU、内存硬件、硬盘以及网络硬件等都是这个系统的成员。

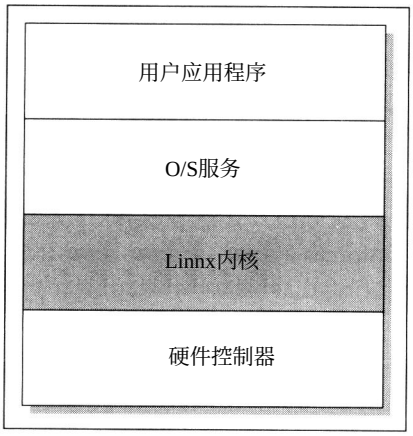

图4-1-1 Linux系统可分为4个子系统

这个系统划分方法是照搬 Garlan和Shaw在[Garlan 1994]中介绍的分层类型。每个子系统层都只能与跟它相邻的层通信。此外，子系统之间的依赖关系是从上到下的：靠上的层依赖于靠下的层，但靠下的层并不依赖于靠上的层。

因为本文的重点是 Linux内核，所以本文将完全不介绍用户应用程序子系统，对硬件和O/S服务也只考虑它们与Linux内核子系统的接口。

1.2 内核的目标

Linux内核向用户进程提供了一个虚拟机器接口。编写进程的时候并不需要知道计算机上安装了哪些物理硬件 —— Linux内核会把所有的硬件抽象成统一的虚拟接口。此外， Linux以对用户透明的方式支持多任务：每个进程工作时就象它是计算机上唯一的进程，好象是独占使用了主存和其他硬件资源一样。内核实际上同时运行许多个进程，并负责对硬件资源的间接访问，这样可以保证各个进程访问的公平性，并保证进程间的安全性。

1.3 内核结构的概述

Linux内核由5个主要的子系统构成：

China-pub.com 下载

1. 进程调度程序（SCHED）负责控制进程访问CPU。调度程序所使用的策略可以保证进程能够公平地访问CPU，同时保证内核可以准时执行一些必需的硬件操作。

2. 内核管理程序（MM）使多个进程可以安全地共享机器的主存系统。此外，内核管理程序支持虚拟内存。虚拟内存使得Linux可以支持进程使用超过系统中的内存数量的内存。暂时用不着的存储信息可以交换出内存，存放到使用文件系统的永久性存储器上，然后在需要它们的时候再交换回来。

3. 虚拟文件系统（VFS）。通过提供一个所有设备的公共文件接口， VFS抽象了不同硬件设备的细节。此外， VFS支持与其他操作系统兼容的不同的文件系统格式。

4. 网络接口（NET）提供了对许多建网标准和网络硬件的访问。

5. 进程间通信（IPC）子系统为单个Linux系统上进程与进程之间的通信提供了一些机制。

图4-1-2是Linux内核的高级分解，图中线段的箭头是从依赖别人的子系统指向被依赖的子系统。

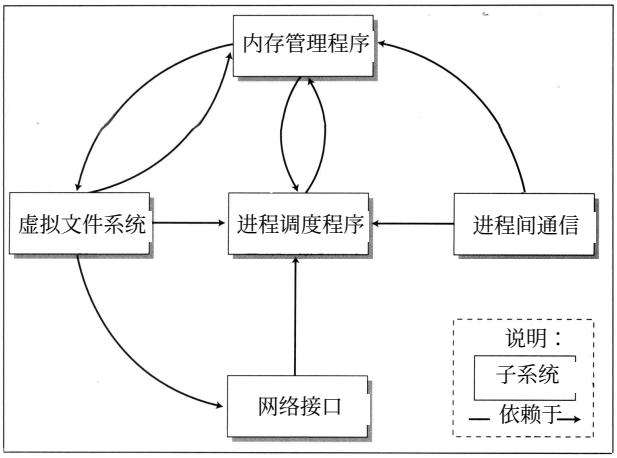

图4-1-2 核心子系统

从图中可以看出，最中心最重要的子系统是进程调度程序：其他所有的子系统都依赖于进程调度程序，这是因为所有的子系统都需要中断和恢复进程的执行。一般来说，子系统会中断那些等待硬件操作完成的进程，同时恢复那些操作已经完成了的进程。例如，当某进程试图通过网络发送消息时，网络接口可能需要中断该进程，直到硬件成功完成了消息的发送。当消息发送完以后（或者硬件返回出错信号），则网络接口将用返回码来恢复该进程。返回码显示了操作是成功完成还是失败。别的子系统（内存管理程序、虚拟文件系统、以及进程间通信）都是由于相似的原因而必须依赖于进程调度程序。

相比之下，其他的依赖关系就不那样明显了，但是它们也同样重要：

- 在进程恢复执行时，进程调度程序将使用内程管理程序来调整硬件内存映射。
- 进程间通信子系统依赖于内存管理程序来支持共享内存通信机制。进程除了可以访问它们通常的私有内存外，共享内存通信机制将使它们可以访问一个公共的内存区。
- 虚拟文件系统使用网络接口来支持网络文件系统（ NFS），它还使用内存管理程序来提供ramdisk设备。
- 内存管理程序使用虚拟文件系统来支持交换。这是内存管理程序之所以依赖于进程调度程序的唯一原因。当某进程访问的内存当前已经被交换出内存时，内存管理程序请求文件系统从永久性存储设备中去取该内存，并中断该进程。

除了显式的依赖关系以外，内核中所有的子系统还依赖于一些在任何子系统中都没有显示出来的公共资源。它们包括：所有内核子系统用于分配和释放内核所使用内存的过程，打

China-pub.com
下载

印警告或错误信息的过程，以及系统调试过程等等。这些资源之所以称之为非显式的，是因为它们只能在图1-1所示的内核层中使用。

这一级别的系统结构类型类似于Garlan和Shaw在[Garlan 1994]中所讨论的数据抽象类型。这里描述的每一个系统都包含状态信息，这些状态信息可以使用过程接口来访问，而每个子系统都需要维护它们所管理的资源的完整性。

1.4 支持多个开发人员

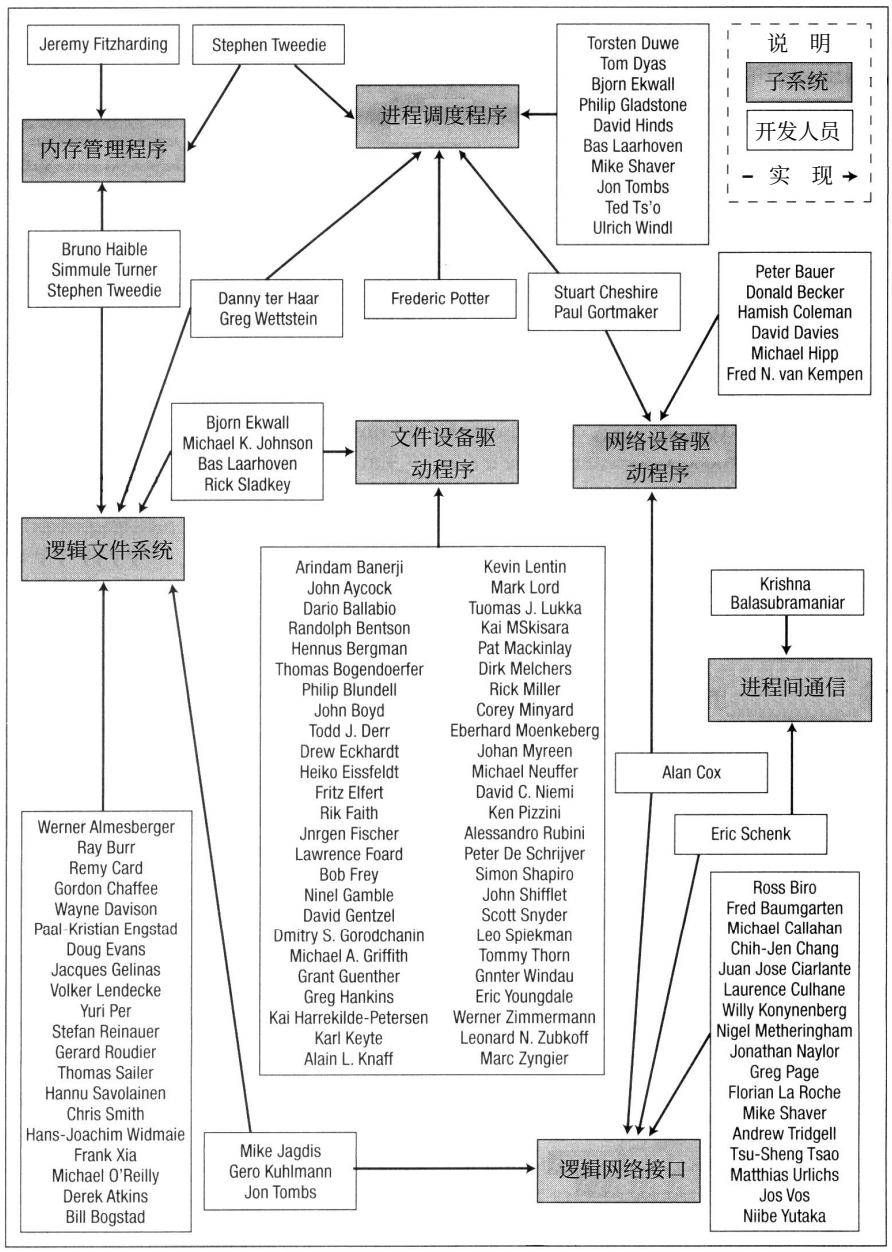

图4-1-3 开发人员职责划分图

China-pub.com 下载

Linux系统是由大量的志愿者所开发的（当前 CREDITS文件列举出了196个曾经进行过Linux系统开发的开发人员）。开发人员非常多，而且他们都是志愿者，这势必会影响到系统的构造。因为开发人员在地域上分布上很广泛，所以很难开发出一个紧耦合的系统来 —— 开发人员常常需要阅读别人的代码。因此， Linux系统系统结构的原则是：把那些可能需要作很大修改的子系统 —— 文件系统、硬件接口和网络系统 —— 设计成高度模块化的系统。例如，一个 Linux实现可能需要支持具有不同接口的许多硬件设备，一种朴实的系统结构是把所有硬件设备的实现做在一个子系统中。而一种支持多个开发人员的更好的方法是：把每个硬件设备的代码分离成一个设备驱动程序，这些驱动程序是文件系统中独立的模块。

图4-1-3列出了曾经在Linux内核开发方面作出了贡献的大部分开发人员，以及他们已经实现了的区域。一些开发人员修改了内核的很多部分。为了描述方便，我们没有列出这些开发人员。例如， Linus Torvalds是大部分内核子系统的实现者，尽管后续的开发是由其他人完成的。该图不能做到十分精确，因为在内核开发过程中，开发人员的组成经常变动，但从图中基本可以看出哪些系统是开发人员花费了大量精力去实现的。

该图重新确认了前面描述过的内核的大规模结构。很有趣的是，很少有开发人员曾经在多个系统上工作过，但这样的人的确存在，它主要发生在那些存在子系统依赖性的多个子系统上。这种组织方式符合 Melvin Conway所提出的拇指规则（参见 [Raymond 1993]），即系统的组织应该反映开发人员的组织。大部分开发人员都在开发硬件设备驱动程序、逻辑文件系统模块、网络设备驱动程序以及网络协议模块。内核的这4个区域的系统结构是最具有扩展性的，这一点也不奇怪。

1.5 系统数据结构

1.5.1 任务列表

进程调度程序为每个活跃的进程维护一块数据。这些数据块是存储在链表中的，该链表称为任务列表。进程调度程序常常维护一个当前指针，该指针显示当前的活跃进程。

1.5.2 内存映射

内存管理程序为每个进程都存储了一个从虚拟地址到物理地址的映射表，它还存储了一些有关如何取和替换特定页面的其他信息。这一信息存放在内存映射数据结构中，该结构存放在进程调度程序的任务列表中。

1.5.3 索引节点

在逻辑文件系统上，虚拟文件系统使用索引节点来表示文件。索引节点数据结构存放着从文件块编号到物理设备地址的映射。如果两个进程打开了同一个文件，则这两个进程可以共享索引节点数据结构。这一共享是通过指向同一个索引节点的任务数据块来完成的。

China-pub.com
下载

1.5.4 数据连接

所有的数据结构都保存在进程调度程序的任务列表中。系统的每个进程都会有一个数据结构，该数据结构中包含着指向内存映射信息的指针，以及指向代表所有打开文件的索引节点的指针。最后，任务数据结构还包含着指向一些数据结构的指针，这些数据结构代表了与每个任务相联系的打开的网络连接。

China-pub.com
下载

第2章　子系统的系统结构

2.1　进程调度程序系统结构

2.1.1　目标

进程调度程序是Linux内核中最重要的子系统。它的目标是控制对计算机 CPU的访问，这里不仅包括用户进程的访问，还包括其他内核子系统的访问。

2.1.2　模块

调度程序可以划分为三个主要的模块：

1) 调度策略模块负责判定哪个进程对 CPU有访问权；策略的设计应该使进程可以公平地访问CPU。

2) 系统结构相关的模块一般使用公共接口设计而成，这样可以抽象出任意特定的计算机系统结构的细节。这些模块负责与 CPU的通信，以中断或者恢复进程的执行。这些操作包括需要为每个进程保留哪些寄存器和状态信息，以及执行汇编代码来影响中断或者恢复操作。

3) 独立于系统结构的模块与策略模块互相通信，来确定接下来执行哪个进程，然后调用系统结构相关的模块来恢复适当的进程。此外，这个模块调用内存管理程序，来确保已经为被恢复的进程准备好了内存硬件，如图 4-2-1所示。

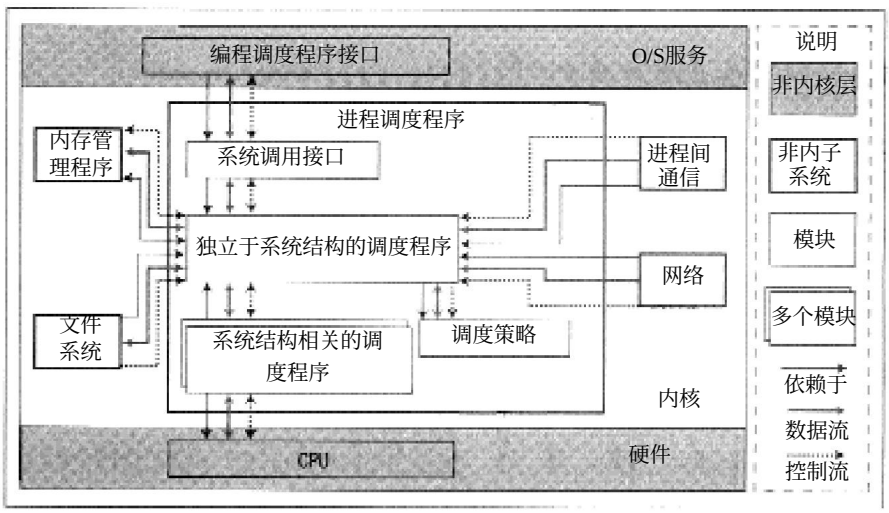

图4-2-1　上下文中的进程调度程度子系统

系统调用接口模块只允许用户进程访问由内核显式打开的那些资源。这就决定了用户进程必须依赖于内核能提供一个很少变动的且定义良好的用户接口，而不管其他的内核模块在

China-pub.com
下载

实现上怎样变化。

2.1.3 数据表达

调度程序维护一个数据结构 —— 任务列表。其中每个项目对应一个活跃进程，这个数据结构包含足够的信息，可以中断和恢复进程的执行，但同时也包含了一些记帐和状态信息。在整个内核层中都可以用到这个数据结构。

2.1.4 依赖性、数据流和控制流

正如前面所说的那样，进程调度程序调用内存管理程序子系统。因此，进程调度程序依赖于内存管理子系统。此外，其他所有的内核子系统也都依赖于进程调度程序，在等待硬件请求完成时，可以中断和恢复进程。这些依赖性是通过函数调用以及对共享任务列表数据结构的访问来体现的。所有的内核子系统都会读写代表当前任务的数据结构，这导致了整个系统中数据的单向流动。

除了内核层中的数据流和控制流以外， O/S服务层为用户进程提供了一个接口，以登录定时器通知。这对应于 [Garlan 1994]中所描述的隐式执行系统结构类型。它会引起从调度程序到用户进程的控制流。类似恢复静止进程这样普通的例子不应看作是普通意义上的控制流，因为用户进程不能检测出这个操作。最后，调度程序与 CPU通信，以便中断和恢复进程，这会导致数据流和控制流。CPU负责中断当前执行的进程，并允许内核调度其他的进程。

2.2 内存管理程序系统结构

2.2.1 目标

内存管理程序子系统负责控制进程对硬件内存资源的访问。这是通过硬件内存管理系统来完成的，该系统提供进程内存引用与计算机的物理内存之间的映射。内存管理程序子系统为每个进程都维护一个这样的映射关系，这样，两个进程就可以访问同一个虚拟内存地址，而实际使用的是不同的物理内存位置。此外，内存管理程序子系统支持交换，它把暂时不使用的内存页面移出内存，存放到永久性存储器中，这样计算机就可以支持比物理内存要多的虚拟内存。

2.2.2 模块

内存管理程序子系统由三个模块组成：

1) 系统结构相关的模块提供了一个内存管理硬件的虚拟接口。

2) 系统结构无关的管理程序执行每个进程的映射工作和虚拟内存的交换工作。当出现页面错时，该模块负责判定应该把哪个内存页面移出内存 —— 这里不存在独立的策略模块，这是因为这个策略不太可能会有变化。

3) 内存管理程序子系统还提供了一个系统调用接口，以提供对用户进程的约束访问。该接口允许用户进程分配和释放存储，并可以执行内存映射文件 I/O。

2.2.3 数据表示

内存管理程序为每个进程存储一个从物理地址到虚拟地址的映射。在进程调度程度的任

China-pub.com 下载

务列表数据结构中，这一映射关系是以引用的方式存储的。除了这一映射关系以外，数据块中的其他细节也将告诉内存管理程序如何取页面及存储页面。例如，可执行代码可以使用可执行映象作为后援存储，但动态分配的数据必须备份到系统分页文件中。最后，内存管理程序在这个数据结构中存储了许可信息和计帐信息，以便保证系统安全性，如图 4-2-2所示。

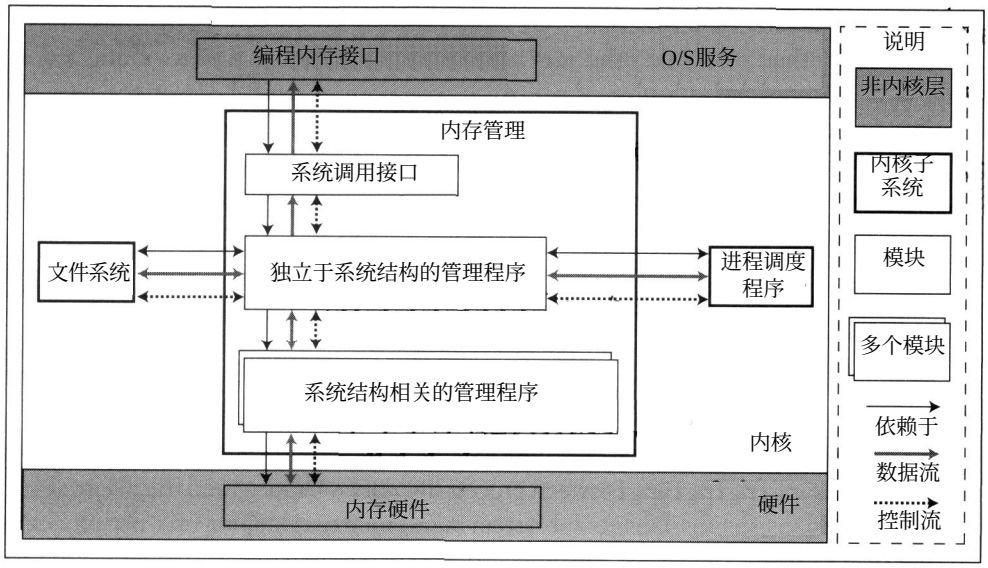

图4-2-2 上下文中的内存管理程序子系统

2.2.4 数据流、控制流和依赖性

上下文中的虚拟文件系统如图 4-2-3所示。

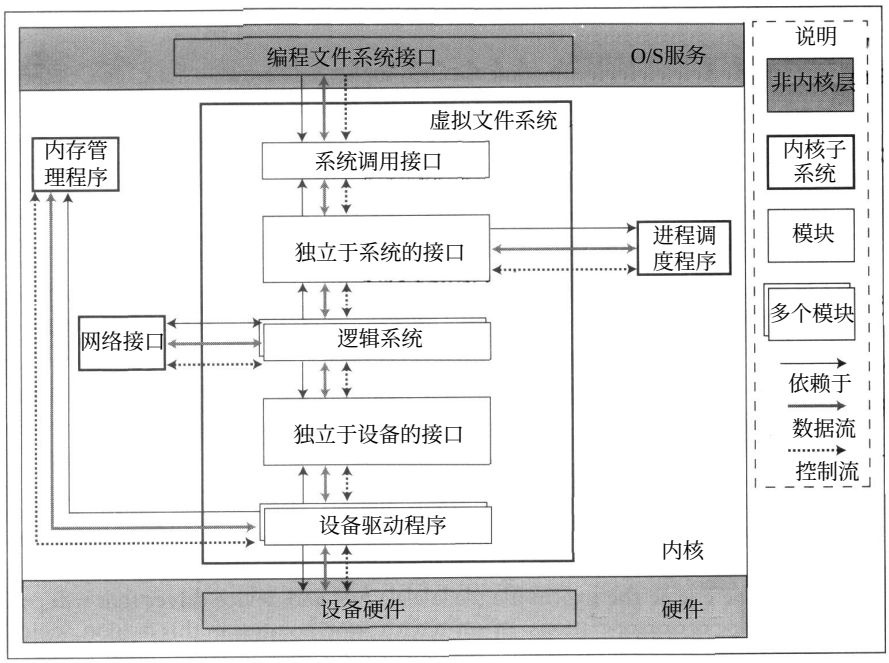

图4-2-3 上下文中的虚拟文件系统

China-pub.com
下载

内存管理程序控制内存硬件，在发生页面错时，它从硬件接收到一个通知 —— 这意味着在内存管理程序模块和内存管理程序硬件之间存在单向的数据流和控制流。此外，内存管理程序使用文件系统来支持交换和内存映射 I/O。这就意味着内存管理程序需要向文件系统作过程调用，以便把页面存储到永久性存储器中，或者从永久性存储器中读取页面。因为文件系统请求不能立即完成，所以内存管理程序需要中断某个进程的执行，直到该内存页面被交换回来。这就导致了内存管理程序必须向进程调度程序作过程调用。此外，因为每个进程的内存映射信息是存储在进程调度程序的数据结构中的，所以在内存管理程序和进程调度程序之间存在单向的数据流。用户进程可以在进程的地址空间中建立起新的内存映射关系，并在新映射的区域登记它们，以便发出页错误通知。这又导致了一个从内存管理程序通过系统调用接口模块到用户进程的控制流。用户进程不会发出传统意义上的数据流，但是用户进程使用系统调用接口模块中的系统调用 select，可以从内存管理程序获取一些信息。

2.3 虚拟文件系统系统结构

2.3.1 目标

虚拟文件系统设计的目的是为了提供一个存储在硬件设备上的数据的统一视图。几乎计算机中的所有设备都是使用通用设备驱动程序接口来表示的。而虚拟文件系统则更进一步，允许系统管理员在任意物理设备上挂装任意一个逻辑文件系统的集合，逻辑文件系统提高了与其他操作系统标准的兼容性，并允许开发人员用不同的策略来实现文件系统。这个虚拟文件系统抽象了物理设备和逻辑文件系统的细节，允许用户进程使用公共接口访问文件，而无需知道文件实际驻留在哪个物理或逻辑系统上。

除了传统的文件系统目标以外，虚拟文件系统还负责装入新的可执行程序。这个任务是由逻辑文件系统模块完成的，这就使得 Linux可以支持许多种可执行文件格式。

2.3.2 模块

1) 每个硬件控制器都对应一个设备驱动程序模块。因为存在大量不兼容的硬件设备，所以设备驱动程序的数量也很多。 Linux系统的大多数通用扩展都是增加了新的设备驱动程序。

2) 独立于设备的接口模块提供了所有设备的统一视图。

3) 每个受支持的文件系统都有一个逻辑文件系统模块。

4) 独立于系统的接口提供了硬件资源的视图，该视图与硬件和逻辑文件系统无关。这一模块使用面向块的或面向字符的文件接口来表示所有资源。

5) 最后，系统调用接口为用户进程提供了对文件系统的受控制的访问。这个虚拟文件系统只把特定的功能导出给用户进程。

2.3.3 数据表示

所有的文件都是使用索引节点来表示的，每个索引结点结构都包含着位置信息，以指定文件块在物理设备上的位置。此外，索引节点存储了指向逻辑文件系统中过程的指针，还存储了指向将要执行所需读写操作的设备驱动程序的指针。通过以这种方式存储函数指针，逻辑文件系统和设备驱动程序在内核中注册自己时，就不需要内核依赖于任何特定的模块。

China-pub.com 下载

2.3.4 数据流、控制流和依赖性

有个特定的设备驱动程序，它就是 ramdisk，这个设备分配主存的一个区域，并把它当作永久性存储设备来对待。这个设备驱动程序使用内存管理程序来完成它的任务，所以在文件系统设备驱动程序和内存管理程序之间存在依赖性，控制流和数据流。

网络文件系统是受支持的一个特定的逻辑文件系统（仅作为一个客户），该文件系统可以访问另一台计算机的文件，就好象它们是本地计算机的一部分。为了完成这个任务，逻辑文件系统模块需要使用网络子系统来完成它的任务。这就导致在两个子系统之间存在依赖性、数据流和控制流。

正如前面在2.2节中所说的那样，内存管理程序使用虚拟文件系统来完成内存交换和内存映射I/O。此外，在进程等待硬件请求完成时，虚拟文件系统使用进程调度程序来中断进程，而一旦请求完成则立刻恢复它们的执行。最后，系统调用接口允许用户进程调用虚拟文件系统，以便存储或者获取数据。与前一个子系统不同的是，这里没有为用户提供隐式调用的注册机制，所以在虚拟文件系统到用户进程之间没有控制流（恢复进程并不认为是控制流）。

2.4 网络接口系统结构

2.4.1 目标

网络子系统允许Linux系统通过网络与其他系统相连接。因为它支持大量的硬件设备，所以相应地也需要使用大量的网络协议。网络子系统抽象了这两者在实现上的细节，这样用户进程和其他内核子系统在访问网络时就无需知道使用的是什么物理设备和协议了。

2.4.2 模块

上下文中的网络接口子系统如图 4-2-4所示。

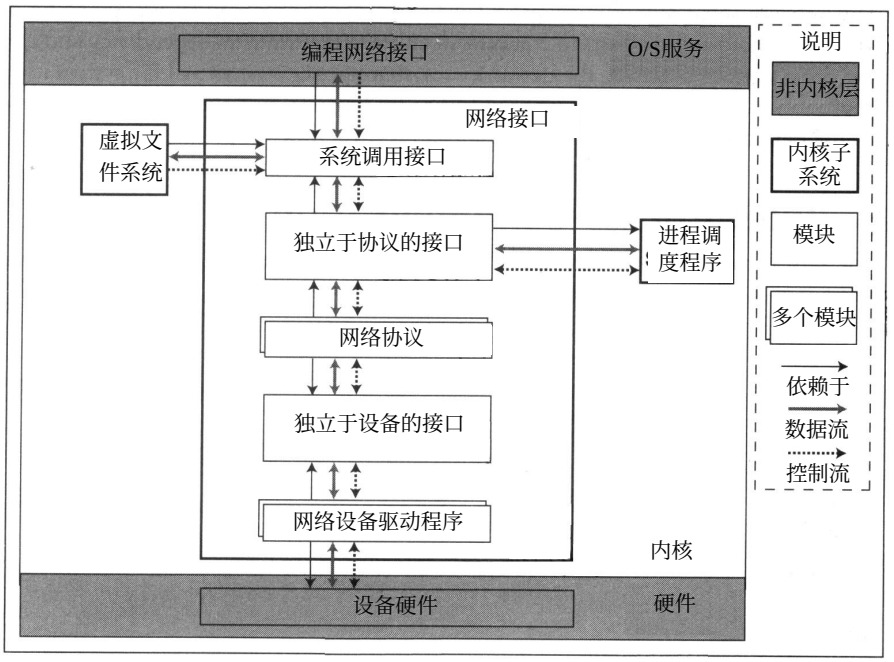

图4-2-4 上下文中的网络接口子系统

China-pub.com
下载

1) 网络设备驱动程序与硬件设备通信。每个可能的硬件设备都对应一个设备驱动程序模块。

2) 独立于设备的接口模块为所有的硬件设备提供了一个统一的视图，这样在子系统的高级别上就不需要关于所使用的硬件的特定知识了。

3) 网络协议模块负责实现每个可能的网络传输协议。

4) 独立于协议的接口模块提供了一个独立于硬件设备和网络协议的接口。其他内核子系统可以通过该接口模块来访问网络，而无需依赖于特定的协议或硬件。

最后，系统调用接口限制了用户进程可以访问的导出过程。

2.4.3 数据表示

每个网络对象都表示成一个套接字。套接字与进程的关系类似于索引节点与进程的关系。通过使两个任务数据结构指向同一个套接字数据结构，进程之间可以共享套接字。

2.4.4 数据流、控制流和依赖性

在等待硬件请求完成时，网络子系统使用进程调度程序中断和恢复进程的执行（这导致了依赖性、控制流和数据流）。此外，网络子系统为虚拟文件系统提供了逻辑文件系统（ NFS）的实现，这导致了虚拟文件系统依赖于网络接口，并与它之间存在数据流和控制流。

2.5 进程间通信系统结构

因为这个子系统不如其他子系统有趣，所以本文不介绍进程间通信子系统的系统结构。

China-pub.com
下载

第3章 结 论

Linux内核是整个Linux系统的系统结构中的一层。在概念上内核是由5个主要的子系统组成的：进程调度程度、内存管理程序、虚拟文件系统、网络接口和进程间通信接口。这些子系统之间是通过函数调用和共享数据结构相互通信的。

在最高级别上，Linux内核的系统结构类型与Garlan和Shaw在[Garlan 1994]中所描述的数据抽象类型十分接近。内核是由子系统所组成的，而子系统通过使用特定的过程接口来保持内部表示的一致性。通过对每个子系统进行认真的解剖分析，我们可以看出它的系统结构类型与Garlan和Shaw的分层类型十分相似。每一个子系统都是由模块组成的，这些模块只能与相邻层相通信。

Linux内核的概念系统结构已经被证明是非常成功的；而它之所以成功，关键在于它规定了开发人员的组织和系统扩展性。Linux内核系统结构必须支持大量独立的志愿开发人员。这就要求投入工作量最大的系统分区（如硬件设备驱动程序和文件以及网络协议）以一种可扩展的方式来实现。Linux系统结构者为了使这些系统可以扩展，使用了一种数据抽象技术：每个硬件设备驱动程序都是作为一个独立的模块来实现的，而这些模块又都支持一个通用接口。通过这种方式，单个的开发人员可以加入一个新的设备驱动程序，并且不需要与 Linux内核的其他开发人员进行很多交互。内核由大量的志愿开发人员实现成功，这个事实已经证明了这个策略的正确性。

Linux内核的另一个重要扩展是加入了更多受支持的硬件平台。通过把与硬件相关的所有代码分离到每个子系统各自的模块中，系统的系统结构支持了这种扩展性。通过这种方式，一小群开发人员通过重新实现内核的机器相关部分，就可以把 Linux内核移植到新的硬件系统结构下。

China-pub.com
下载

附录A 术语定义

设备驱动程序（Device Driver）

设备驱动程序是与特定的硬件设备交互所需要的所有代码。设备驱动程序实际上是内核的一部分，但是Linux内核提供了一种机制，它允许动态装入设备驱动程序。

索引节点（I-Node）

索引节点即index node，可以被文件系统用来跟踪文件系统数据所存储的硬件地址。每个索引节点存储了一个文件块到物理块的映射关系，以及安全性方面以及计帐方面的其他一些信息。

网络文件系统（Network File System, NFS）

网络文件系统是一个文件系统接口，它把远程计算机上存储的文件表示成本地计算机上的文件系统。

进程（Process）

进程（也称为任务）就是执行中的程序，它由可执行代码和动态数据所组成。

内核为每个进程都保留了足够多的信息，以便停止或恢复它的执行。

Ramdisk

Ramdisk是一个设备驱动程序，它把主存的一块区域用作文件系统设备。它允许把经常访问的文件存储在能随时提供有效访问的区域中，在使用 Linux支持硬实时的需求时，这个特征非常有用。对于一般的情况而言，普通的文件系统高速缓存机制将高效地使用内存，以提供对文件的高效访问。

交换

Linux支持进程使用超过计算机上物理内存数量的内存。要做到这一点，内存管理程序必须把暂时不使用的内存页面交换到永久性存储设备中。当以后访问该内存时，再把它交换回主存（这时可能会导致其他页面被交换出去）。

任务（Task）

参见进程。

China-pub.com
下载

附录B 参考文献

Garlan 1994

David Garlan和Mary Shaw, An Introduction to Software Architecture, Advances in Software Engineering and Knowledge Engineering, 第1卷, World Scientific Publishing Company, 1993。

Monroe 1997

Robert T.Monroe, Andrew Kompanek, Ralph Melton和David Garlan Architectural Styles, Design Patterns and Objects, IEEE Software January 1997, pp43-52。

Paker 1997

Slackware Linux Unleased, Timothy Parker等人所著, Sams Publishing, 201 West 103rd Street, Indianapolis。

Perry 1992

Dewayne E.Perry和Alexander L.Wolf, Foundations for the Study of Software Architecture, ACM SIGSOFT Software Engineering Notes, 17:4, October 1992 pp.40-52。

Raymond 1993

The New hackers Dictionary, 第2版, 由Eric S.Raymond编辑, The MIT Press, Cambridge Massachusetts, 1993。

Rusling 1997

The Linux Kernel David A.Rusling所著, 初稿, 版本 0.1-13(19), ftp://sunsite. unc.edu /pub/Linux/docs/linux-doc-project/linux-kernel/或者 http: //www. linuxhq. com/guides /TLK/index.html。

Soni 1995

Soni,D.; Nord, R.L.; Hofmeister, C., Software Architecture in Industrial Applications, IEEE ICSE 1995, pp.196-210。

Tanenbaum 1992

Modern Operating Systems, Andrew S.Tanenbaum所著, Prentice Hall, 1992。

Wirzenius 1997

Lars Wirzenius所著Linux System Administrators ' Guide 0.6, http://www.iki.fi/liw /linux/sag/ 或者 http://www.linuxhq.com/LDP/LDP/sag/index.html。